Universitat de Vic
Escola Politècnica Superior

Disseny i muntatge d'un mòdul d'entrades digitals per comunicació modbus.

Joan Barniol i Noguer

Programes Microcontrolador:

- **Programa Test_PLACA_Pmicro.c**
- **Programa Principal_PLACA_Pmicro.c**
- **Protocol_MODBUS_PLACA_Pmicro.c**

Programa Test_PLACA_Pmicro.c

```
////////////////////////////////////////////////////////////////////////////////////
////                          Programa Test_PLACA_Pmicro.c                      ////
////                                                                            ////
////    Treball Final de Carrera                                                ////
////                                                                            ////
////    Disseny i muntatge d'un mòdul d'entrades digitals per comunicació Modbus.  ////
////                                                                            ////
////                          -->> Joan Barniol i Noguer <<--                   ////
////                                                                            ////
////    Director: Moises Serra Serra                                            ////
////                                                                  Juny 08   ////
////                                                                            ////
////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////
//*********************************************************************************//
// Aquest programa, ha estat desenvolupat per comprobar el funcionament del Hardware    //
// del mòdul d'entrades digitals. COMPROVACIÓ: PIC, LED, RELES.                     //
//*********************************************************************************//
////////////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////////////
//*********************************************************************************//
//*******************************                   *******************************//
//*******************************      CONFIGURACIONS     *******************************//
//*******************************                   *******************************//
//*********************************************************************************//
////////////////////////////////////////////////////////////////////////////////////
//                                                                                //
// HS : hight speed (tipus d'oscil·lador utilitzat)                               //
// NOWDT : No watch dog time                                                      //
// NOLVP : N low voltage programing                                               //
// NOBROWNOUT: Brown out : per sota la V aliment. el micro fa un rset. Vmin funcionament//
// NOPROTECT: si ens conectem al micro grabat de la placa, el podrem llegir el programa //
// PUT: Power up Timer                                                            //
//                                                                                //
////////////////////////////////////////////////////////////////////////////////////

#include    <18F4520.h>
#use        delay(clock=11059200)
#fuses      HS, NOWDT, NOLVP, NOBROWNOUT, NOPROTECT, PUT


////////////////////////////////////////////////////////////////////////////////////
//                                                                                //
// Aquets #define, serviran per engegar o parar els dos Reles i els leds que actuen     //
// com a outputs                                                                  //
//                                                                                //
////////////////////////////////////////////////////////////////////////////////////

#define     refresc_entrades 30         // posa 30 cada vegada que escribim refresc_entrades
#define     Rele_K1_on      output_high(PIN_D6);
#define     Rele_K2_on      output_high(PIN_D7);
#define     Rele_K1_off     output_low (PIN_D6);
#define     Rele_K2_off     output_low (PIN_D7);
#define     led_15_on       output_high(PIN_E1);
#define     led_15_off      output_low (PIN_E1);
#define     led_14_on       output_high(PIN_E2);
#define     led_14_off      output_low (PIN_E2);
#define     led_16_on       output_high(PIN_B6);
#define     led_16_off      output_low (PIN_B6);
#define     k_ref_perif     20



////////////////////////////////////////////////////////////////////////////////////
//*********************************************************************************//
//*******************************                   *******************************//
//*******************************  DEFINICIÓ DE VARIABLES  *******************************//
//*******************************                   *******************************//
//*********************************************************************************//
////////////////////////////////////////////////////////////////////////////////////

    byte ref_perif;         //Definim aquesta variable que servirà per fer la interrupció lenta
    byte sortides;          //El byte sortides és una variable que té tots els bits definits
        #bit    int_rapida  =sortides.0
        #bit    int_lenta   =sortides.1
        #bit    no1         =sortides.2
        #bit    no2         =sortides.3
        #bit    no3         =sortides.4
```

```
        #bit    no4         =sortides.5
        #bit    no5         =sortides.6
        #bit    estat_led   =sortides.7

////////////////////////////////////////////////////////////////////////////////
//****************************************************************************//
//*****************************                      *****************************//
//*****************************       CAPÇALERES     *****************************//
//*****************************                      *****************************//
//****************************************************************************//
////////////////////////////////////////////////////////////////////////////////

void    inicialitzar    (void);  // Definir totes les entrades/sortides, inicialitzar variables
void    reles           (void);  // Activar relés



////////////////////////////////////////////////////////////////////////////////
//****************************************************************************//
//*****************************                      *****************************//
//*****************************  RUTINES D'INTERRUPCIÓ *****************************//
//*****************************                      *****************************//
//****************************************************************************//
////////////////////////////////////////////////////////////////////////////////

#INT_TIMER1                     // Envia el programa a la posció de la interrupció
void timer_interrupt (void)     // Amb aquesta funció aconseguim una int.ràpida a uns 5,5ms i una
                                // lenta cada k_ref_perif vegades d'interrupció ràpida (20 vegades)
{
    set_timer1(50175);          // Un "set_timer1" inicialitza el timer 1 com a contador
(desbordament)
    int_rapida=1;               // tenim una interrupció ràpida.
    if (!ref_perif)             // Quan ref perif arribi a 0 entra a fer la interrupció lenta
    {   int_lenta = !int_lenta; // canvia l'estat de l'interrupció lenta 0-1 o 1-0
        ref_perif = k_ref_perif; // en el byte ref_perif i recarrego el valor de la constant k_ref_perif
    }
    ref_perif--;                // Decrementem la variable des de el valor K_ref_perif fins a 0
}



////////////////////////////////////////////////////////////////////////////////
//****************************************************************************//
//*****************************                      *****************************//
//*****************************   PROGRAMA PRINCIPAL  *****************************//
//*****************************                      *****************************//
//****************************************************************************//
////////////////////////////////////////////////////////////////////////////////

void main()
{
    inicialitzar();                 // utilitzem la funcio d'inicialitzar ports.

    set_tris_c(0x98);               // configurem port C

    while(TRUE)                     // bucle principal (repetitiu)
    {
        if (int_lenta)              // k_ref_perif vegades una interrupció ràpida
        {
            int_lenta=0;            // Posem la interrupcio lenta a 0.
                                    // Passades 20 ràpides es torna a activar
            if (estat_led)  led_14_on   // si estat led és 1 engeguem el led
            if (!estat_led) led_14_off  // si estat led és 0 parem el led

            estat_led = !estat_led;     // canviem el valor de estat led per la seguent passada

            if (input(pin_B0))          // podem comprovar que les sortides de rele també funcionen
                rele_K2_on
            else
                rele_K2_off
        }
    }
}
```

```
////////////////////////////////////////////////////////////////////////////////////
//**********************************************************************************//
//******************************                  ******************************//
//******************************       INICIALITZAR       ******************************//
//******************************                  ******************************//
//**********************************************************************************//
////////////////////////////////////////////////////////////////////////////////////

void inicialitzar (void)
{
//Configuració de Ports
    set_tris_a(0x3f);           // Bloc de leds 2, configurat com a entrades
    OUTPUT_a(0);                // OUTPUT_x(value) col·loca el valor "value" al lloc indicat x
    set_tris_b(0x3F);           // Sortida led, entrada Switch, entrada switch S_int
    OUTPUT_b(0);
    set_tris_c(0x98);           //s16,s15,,sda,scl,trans
    OUTPUT_c(0);
    set_tris_d(0x3f);           //sortides reles, bloc leds 1
    OUTPUT_d(0);
    set_tris_e(0x00);           //s20, 2 leds
    OUTPUT_e(0);

// Aquestes sortides estan definides al principi del programa quins pins pertanyen
    led_14_off                  // Les posem totes 3 a 0 per iniciar el programa PIN E2
    led_15_off                  // PIN E1
    led_16_off                  // PIN B6
    estat_led = 0;              // Variable que quan sigui 1 fara Dled14=ON i quan 0 Dled14=OFF

//Inicialitza perifèrics (per evitar problemes els deshabilitem)
    setup_adc(ADC_OFF);                     // instrucció   setup_adc(mode)si tenim OFF inhabilita A/D
    setup_ccp1(CCP_OFF);                    // inhabilita ccp1
    setup_ccp2(CCP_OFF);                    // inhabilita ccp2
    setup_timer_1(T1_INTERNAL|T1_DIV_BY_1); // Refresc de programa
                                            // T1 Internal- Sets interval clock or source
                                            // T1_Div_By_1- Prescale as 1

//Timers
    set_timer1(50175);                      // Carrega el valor (per els 5,5m)
    enable_interrupts(INT_TIMER1);          // Permet interrupció TIM1
    enable_interrupts(GLOBAL);              // Aquesta l'hem de permetre per poder tenir habilitada la
IN_TIM1
}




        ////////////////////////////////////////////////////////////////////////////////////
        //**********************************************************************************//
        //**********************************************************************TFC.1_V2//
        ////////////////////////////////////////////////////////////////////////////////////
```

Programa Principal_PLACA_Pmicro.c

```
////////////////////////////////////////////////////////////////////////////////////////
////                        Programa Principal_PLACA_Pmicro.c                       ////
////                                                                                ////
////    Treball Final de Carrera                                                    ////
////                                                                                ////
////    Disseny i muntatge d'un mòdul d'entrades digitals per comunicació Modbus.   ////
////                                                                                ////
////                           -->> Joan Barniol i Noguer <<--                      ////
////                                                                                ////
////    Director: Moises Serra Serra                                                ////
////                                                                  Juny 08       ////
////                                                                                ////
////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////
//************************************************************************************//
// Aquest és el programa principal del projecte, gestiona la lectura d'entrades i     //
// realitza la comunicació Modbus.                                                    //
//************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////////////////
//************************************************************************************//
//*******************************                  *******************************//
//*******************************      CONFIGURACIONS      *******************************//
//*******************************                  *******************************//
//************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////
//                                                                                    //
// HS : hight speed (tipus d'oscil·lador utilitzat)                                   //
// NOWDT : No watch dog time                                                          //
// NOLVP : N low voltage programing                                                   //
// NOBROWNOUT: Brown out : per sota la V aliment. el micro fa un rset. Vmin funcionament//
// NOPROTECT: si ens conectem al micro grabat de la placa, el podrem llegir el programa //
// PUT: Power up Timer                                                                //
//                                                                                    //
////////////////////////////////////////////////////////////////////////////////////////

#include    <18F4520.h>                                         // pic que utiltizem
#use        delay(clock=11059200)                               // freq.
#fuses      HS, NOWDT, NOLVP, NOBROWNOUT, NOPROTECT, PUT


////////////////////////////////////////////////////////////////////////////////////////
//                                                                                    //
// Aquets #define, serviran per engegar o parar els dos Reles i els leds que actuen     //
// com a outputs                                                                      //
//                                                                                    //
////////////////////////////////////////////////////////////////////////////////////////

#define     Rele_K1_on      output_high(PIN_D6);
#define     Rele_K2_on      output_high(PIN_D7);
#define     Rele_K1_off     output_low (PIN_D6);
#define     Rele_K2_off     output_low (PIN_D7);
#define     led_15_on       output_high(PIN_E1);
#define     led_15_off      output_low (PIN_E1);
#define     led_14_on       output_high(PIN_E2);
#define     led_14_off      output_low (PIN_E2);
#define     led_16_on       output_high(PIN_B6);
#define     led_16_off      output_low (PIN_B6);
#define     k_ref_perif     20

#define     refresc_entrades 30                     // K_posa 30 quan escribim refresc_entrades
#define     MODBUS_SERIAL_RX_BUFFER_SIZE 64         // Config. MOdbus, tamany, vel. baud rates
#define     MODBUS_SERIAL_BAUD 9600
#define     USE_WITH_PC 1
#define     MODBUS_SERIAL_INT_SOURCE MODBUS_INT_RDA  // interrupcions arriben Usart
```

```
////////////////////////////////////////////////////////////////////////////////////////
//************************************************************************************//
//*******************************                    *******************************//
//*******************************  DEFINICIÓ DE VARIABLES  *******************************//
//*******************************                    *******************************//
//************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////

//definim bytes
int8 coils = 0b00000101;        // 0b li entrem num binari
int8 inputs = 0b00001001;       // Posem uns valors per saber si s'han inicialitzat o no

////////////////////////////////////////////////////////////////////////////////////////
                                                                                    //
// el que es defineix a continuació són 2 registres de 8 words d'on escrivim i          //
// llegim el que es vol enviar per Modbus.Només utilitzem els words 1,2 i 3 per         //
// els dos blocs d'entrades i per la sortida. També els hi donem valors per poder       //
// veure si varien o no.                                                                //
////////////////////////////////////////////////////////////////////////////////////////

//---------------------->>  res   ent1   ent2   rele    adr    res    res   res    <<---//
    int16 hold_regs[]  = {0x8800,0x1500,0x1000,0x0000,0x4400,0x3300,0x2200,0x1100};
    int16 input_regs[] = {0x1100,0x2200,0x3300,0x4400,0x5500,0x6600,0x7700,0x8800};
    int16 event_count  = 0;

    byte mod_adr;                           //switch adreçament modbus
    byte leds1;                             //entrades bloc entrades 1
    byte leds1a;                            //auxiliar bloc entrades 1 antirebot
    byte leds2;                             //entrades bloc entrades 2
    byte leds2a;                            //auxiliar bloc entrades 2 antirebot

    byte ref_perif;                         // Variable que utlitzo per fer la interrupció lenta

    byte sortides;                          // El byte sortides té tots els bits definits
        #bit    int_rapida  =sortides.0     // Farem una interrupció cada 5,5 ms de temps
        #bit    int_lenta   =sortides.1     // Cada unes 20 interrupcions ràpides en tindrem una lenta
        #bit    no1         =sortides.2
        #bit    no2         =sortides.3
        #bit    no3         =sortides.4
        #bit    no4         =sortides.5
        #bit    ini_trama   =sortides.6     // bit de inici
        #bit    fi_trama    =sortides.7     // bit de final


////////////////////////////////////////////////////////////////////////////////////////
//************************************************************************************//
//*******************************                    *******************************//
//*******************************       CAPÇALERES       *******************************//
//*******************************                    *******************************//
//************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////

#include <s_modbus1.c>

void    inicialitzar            (void);     // Definir totes les entrades/sortides, inicialitzar
variables
void    lectura_adr             (void);     // Llegeix l'adraçement del switch
void    lectura_ent_bloc_1      (void);     // Llegeix el les entrades digitals
void    lectura_ent_bloc_2      (void);     // Llegeix el les entrades digitals
void    reles                   (void);     // Activació relés


////////////////////////////////////////////////////////////////////////////////////////
//************************************************************************************//
//*******************************                    *******************************//
//*******************************  RUTINES D'INTERRUPCIÓ  *******************************//
//*******************************                    *******************************//
//************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////

#INT_TIMER1                         // Envia el programa a la posció de la interrupció
void timer_interrupt (void)
{
    set_timer1(50175);              // Un "set_timer1" inicialitza el timer 1 com a contador.
    if (!ref_perif)                 // Quan ref perif arribi a 0 entra a fer la interrupció lenta
    {   int_lenta = !int_lenta;     // canvia l'estat de l'interrupció lenta 0-1
        ref_perif = k_ref_perif;    // en el byte ref_perif i recarrego el valor de la constant
k_ref_perif
    }
    ref_perif--;                    // Decrementem la variable des de el valor K_ref_perif fins a 0
}
```

```
////////////////////////////////////////////////////////////////////////////////
//*****************************************************************************//
//*****************************                   *****************************//
//*****************************    PROGRAMA PRINCIPAL    *****************************//
//*****************************                   *****************************//
//*****************************************************************************//
////////////////////////////////////////////////////////////////////////////////

void main()
{
    inicialitzar();                 // Funció d'inicialitzar definida després del programa main
    modbus_init();                  // funcio que configura el Max en mode de lectura. Està definida a
S_modbus
    set_tris_c(0x9A);               // s16,s15,s19,sda,scl,trans,s18,s19
    lectura_ent_bloc_1();           // Funció de lectura entrades 1, definida després del main
    lectura_ent_bloc_2();           // Funció de lectura entrades 2, definida després del main

    while(TRUE)                     // bucle principal (repetitiu)
    {
        if (int_lenta)              // k_ref_perif vegades una interrupció ràpida
        {
            int_lenta=0;            // Canvio estat interrupció lenta 1-0 abans he fet x= !x;

            lectura_ent_bloc_1();   // Mirem entrades per si estem reben alguna cosa
            lectura_ent_bloc_2();
            reles ();               // funcio per controlar els leds


        }
        if(modbus_kbhit())          // Mirem modbus per si rebem alguna trama
        {   led_15_on               // El Led 15 indica que s'ha rebut una nova trama
            interpret_modbus();     // Funció Modbus definida a S_modbus
            delay_ms(10);
            led_15_off
            led_14_off
        }
    }
}


////////////////////////////////////////////////////////////////////////////////
//*****************************************************************************//
//*****************************                   *****************************//
//*****************************       INICIALITZAR       *****************************//
//*****************************                   *****************************//
//*****************************************************************************//
////////////////////////////////////////////////////////////////////////////////

void inicialitzar (void)
{
//TRIS:assigno entrades i sortides: 0 sortides,1 entrades

//ports
    set_tris_a(0x3f);       // Bloc de leds 2, configurat com a entrades
    OUTPUT_a(0);            // Un OUTPUT_x(value) col·loca el valor "value" al lloc indicat per x
    set_tris_b(0x3F);       // Sortida led, entrada microswitch multiplexats, entrada switch interrupció
    OUTPUT_b(0);
    set_tris_c(0x98);       // s16,s15,s19,sda,scl,trans,s18,s19
    OUTPUT_c(0);
    set_tris_d(0x3f);       // Sortides reles, bloc leds 1
    OUTPUT_d(0);
    set_tris_e(0x00);       // s20, 2 leds
    OUTPUT_e(0);

    leds1=0;                // Posa els Bytes del bloc d'entrades 1 i 2 i els de l'antirebot a 0.
    leds1a=0;               //
    leds2=0;                //
    leds2a=0;               //
    mod_adr=0;              // Dona valor 0 a la variable adreçament
    led_14_off              // Apaga led 14 i 15 (son sortides)
    led_15_off              //

//Inicialitza perifèrics
    setup_adc(ADC_OFF);                     // Inhabilita  el conversor encara que no el fem servir
    setup_ccp1(CCP_OFF);                    // Inhabilita ccp1
    setup_ccp2(CCP_OFF);                    // Inhabilita ccp2
    setup_timer_1(T1_INTERNAL|T1_DIV_BY_1); // Refresc de programa habilita o deshabilita timer 1
```

```
//Timers
    set_timer1(50175);              // Inicialitza el Timer 1 coma contador
    enable_interrupts(INT_TIMER1);  // Permet interrupció del timer 1
    enable_interrupts(GLOBAL);      // Aquest ha d'estar habilitat per poder tenir qualsevol interrupció

    lectura_adr ();                 // realitza la funció de llegir els valors del switch
}


////////////////////////////////////////////////////////////////////////////////
//****************************************************************************//
//*****************************                      *****************************//
//*****************************      LECTURA SWITCH      *****************************//
//*****************************                      *****************************//
//****************************************************************************//
////////////////////////////////////////////////////////////////////////////////

void    lectura_adr     (void)      // Llegir switch 1.0.
{
    mod_adr = 0;                    //Posem el registre a 0

    if (!input(PIN_B5))         bit_set(mod_adr,0);  //Mira si la entrada està a 0
    if (!input(PIN_B4))         bit_set(mod_adr,1);  //Si esta a 0 posem un 1 al bit ADR pertinent
    if (!input(PIN_B3))         bit_set(mod_adr,2);
    if (!input(PIN_B2))         bit_set(mod_adr,3);
    if (!input(PIN_B1))         bit_set(mod_adr,4);

    hold_regs[4] = ((mod_adr + 20)<<8);     // Guardem els Adr en registre Hregs word 4
                                            // 1Word = 2Bytes funcionen desplaçats!!
                                            // Per això afegim "<<8" Mou bits 8 posicions
}


////////////////////////////////////////////////////////////////////////////////
//****************************************************************************//
//*****************************                      *****************************//
//*****************************     LECTURA ENTRADES     *****************************//
//*****************************                      *****************************//
//****************************************************************************//
////////////////////////////////////////////////////////////////////////////////

//llegeix les entrades dels ports i posa un 1 la posició del port on està llegint
//lectura d'entrades del port D  //bloc leds 1//
// Per llegir aquestes 12 entrades vigilem que no hi hagi rebots per això
// Guardem la dada anterior sempre per poder comparar (leds1a).

void lectura_ent_bloc_1 (void)
{
    int16 aux;

    // Si l'entrada conicideix amb el bit ledsanterior guarda valor al bit leds
    if(input(PIN_D5) == bit_test(leds1a,5))             // Si l'entrada d5 es igual al bit 0 de leds1a
    {   if (bit_test(leds1a,5))     bit_set(leds1,5);   // Si el bit leds1a=1 posem un 1 al bit  de
leds1
        else                        bit_clear(leds1,5); // Si es 0 posem un 0 al bit de leds1
    }
    // si l'entrada no és igual a ledanterior, acutalitza el valor a leds anterior
    if (input(PIN_D5)==1)           bit_set(leds1a,5);  //actualitza registre leds1a amb el valor que
te l'entrada
    else                            bit_clear(leds1a,5);

    // repetim això per les 12 entrades

    if(input(PIN_D4) == bit_test(leds1a,4))
    {   if (bit_test(leds1a,4))     bit_set(leds1,4);
        else                        bit_clear(leds1,4);
    }
    if (input(PIN_D4)==1)           bit_set(leds1a,4);
    else                            bit_clear(leds1a,4);

    if(input(PIN_D3) == bit_test(leds1a,3))
    {   if (bit_test(leds1a,3))     bit_set(leds1,3);
        else                        bit_clear(leds1,3);
    }
    if (input(PIN_D3)==1)           bit_set(leds1a,3);
    else                            bit_clear(leds1a,3);

    if(input(PIN_D2) == bit_test(leds1a,2))
    {   if (bit_test(leds1a,2))     bit_set(leds1,2);
        else                        bit_clear(leds1,2);
    }
    if (input(PIN_D2)==1)           bit_set(leds1a,2);
```

```
    else                            bit_clear(leds1a,2);

    if(input(PIN_D1) == bit_test(leds1a,1))
    {   if (bit_test(leds1a,1))     bit_set(leds1,1);
        else                        bit_clear(leds1,1);
    }
    if (input(PIN_D1)==1)           bit_set(leds1a,1);
    else                            bit_clear(leds1a,1);

    if(input(PIN_D0) == bit_test(leds1a,0))
    {   if (bit_test(leds1a,0))     bit_set(leds1,0);
        else                        bit_clear(leds1,0);
    }
    if (input(PIN_D0)==1)           bit_set(leds1a,0);
    else                            bit_clear(leds1a,0);

    aux = ~leds1;       // Fem el complement a 1; exemple: 0000 0001  passa a 1111 1110
    aux = aux<<8;       // Exemple: xxxx xxxx 1111 1110 aquesta funció ho gira i queda 1111 1110 xxxx
xxxx
    aux = aux & 0x3F00; // Aux & 0011 1111 0000 0000; Ex: 1111 1110 xxxx xxxx*0011 1111 0000 0000=0011
1110 0000 0000
                        // amb aquesta operació ens quedem només amb els valors que ens interessa ( 6
entrades )

    hold_regs[1] = aux; // La varialbe aux no és global i per això guardem el valor de aux en el
registre
                        // hold regs word 1 (bloc optos 1). D'aquest word es llegirant les trames (hold
regs 1)
}

void lectura_ent_bloc_2 (void)
{
    int16 aux;

    if(input(PIN_A5) == bit_test(leds2a,5))
    {   if (bit_test(leds2a,5))     bit_set(leds2,5);
        else                        bit_clear(leds2,5);
    }
    if (input(PIN_A5)==1)           bit_set(leds2a,5);
    else                            bit_clear(leds2a,5);

    if(input(PIN_A4) == bit_test(leds2a,4))
    {   if (bit_test(leds2a,4))     bit_set(leds2,4);
        else                        bit_clear(leds2,4);
    }
    if (input(PIN_A4)==1)           bit_set(leds2a,4);
    else                            bit_clear(leds2a,4);

    if(input(PIN_A3) == bit_test(leds2a,3))
    {   if (bit_test(leds2a,3))     bit_set(leds2,3);
        else                        bit_clear(leds2,3);
    }
    if (input(PIN_A3)==1)           bit_set(leds2a,3);
    else                            bit_clear(leds2a,3);

    if(input(PIN_A2) == bit_test(leds2a,2))
    {   if (bit_test(leds2a,2))     bit_set(leds2,2);
        else                        bit_clear(leds2,2);
    }
    if (input(PIN_A2)==1)           bit_set(leds2a,2);
    else                            bit_clear(leds2a,2);

    if(input(PIN_A1) == bit_test(leds2a,1))
    {   if (bit_test(leds2a,1))     bit_set(leds2,1);
        else                        bit_clear(leds2,1);
    }
    if (input(PIN_A1)==1)           bit_set(leds2a,1);
    else                            bit_clear(leds2a,1);

    if(input(PIN_A0) == bit_test(leds2a,0))
    {   if (bit_test(leds2a,0))     bit_set(leds2,0);
        else                        bit_clear(leds2,0);
    }
    if (input(PIN_A0)==1)           bit_set(leds2a,0);
    else                            bit_clear(leds2a,0);

    aux = ~leds2;
    aux = aux<<8;
    aux = aux & 0x3F00;

    hold_regs[2] = aux;
}
```

```
////////////////////////////////////////////////////////////////////////////////////////
//************************************************************************************//
//*******************************                  *******************************//
//*******************************      RELES       *******************************//
//*******************************                  *******************************//
//************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////
//                                                                                    //
//  Si word 3 del registre HOLD REGS els bits 8 o 9 = 1 s'hauran d'engegar els reles   //
//                                                                                    //
////////////////////////////////////////////////////////////////////////////////////////

void reles (void)
{

    if(bit_test(hold_regs[3],8))
        Rele_K1_on
    else
        Rele_K1_off

    if(bit_test(hold_regs[3],9))
        Rele_K2_on
    else
        Rele_K2_off
}

////////////////////////////////////////////////////////////////////////////////////////
//************************************************************************************//
//*********************************************************************************TFC.2_V4//
////////////////////////////////////////////////////////////////////////////////////////
```

Protocol_MODBUS_PLACA_Pmicro.c

```
////////////////////////////////////////////////////////////////////////////////////
////                          Protocol_MODBUS_PLACA_Pmicro.c                    ////
////                                                                            ////
////    Treball Final de Carrera                                                ////
////                                                                            ////
////    Disseny i muntatge d'un mòdul d'entrades digitals per comunicació Modbus.  ////
////                                                                            ////
////                          -->> Joan Barniol i Noguer <<--                   ////
////                                                                            ////
////    Director: Moises Serra Serra                                            ////
////                                                                 Juny 08    ////
////                                                                            ////
////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////
//*********************************************************************************//
// Aquest és el Protocol Modbus que s'ha implementat en el sistema. Es realitzen totes  //
// les funcions necessaries per realitzar la comunicacio amb el mestre                  //
//*********************************************************************************//
////////////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////////////
////                                                                            ////
////  DEFINES:     TOTS AQUETS ELS TENIM DEFINITS EN EL PROG. PRINCIPAL          ////
////                                                                            ////
////  MODBUS_SERIAL_BAUD            Baud Rate definit al prg main a 9600        ////
////  MODBUS_SERIAL_RX_PIN          Valid pin for serial receive                ////
////  MODBUS_SERIAL_TX_PIN          Valid pin for serial transmit               ////
////  MODBUS_SERIAL_RX_ENABLE       Valid pin for serial rcv enable, rs485 only ////
////  MODBUS_SERAIL_RX_BUFFER_SIZE  Tamany memoria definit al prg main a 64     ////
////                                                                            ////
////                                                                            ////
//// SHARED API:  FUNCIONS COMPARTIDES MESTRE ESCLAU                            ////
////                                                                            ////
////  modbus_init()                                                             ////
////    - Inicialitza la comunicació modbus.                                    ////
////                                                                            ////
////  modbus_serial_send_start(address,func)                                    ////
////    - Configura sistema per enviar.  Un cop s'hagi cridat aquesta funció es poden ////
////      enviar dades amb la funció modbus_serial_putc().                      ////
////                                                                            ////
////  modbus_serial_send_stop()                                                 ////
////    - S'ha d'utilitzar per finalitzar els enviaments.                       ////
////                                                                            ////
////  modbus_kbhit()                                                            ////
////    - S'utilitza per comprobar si hem rebut una trama.                      ////
////                                                                            ////
////                                                                            ////
////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
////                                                                            ////
//// Slave API:  FUNCIONS DE L'ESCLAU    (CODIS)                                ////
////                                                                            ////
////  CODI 0x01                                                                 ////
////  void modbus_read_coils_rsp(address,byte_count,*coil_data)                 ////
////    - Wrapper to respond to 0x01(read coils) in the MODBUS specification.   ////
////                                                                            ////
////  CODI 0x02                                                                 ////
////  void modbus_read_discrete_input_rsp(address,byte_count,*input_data)       ////
////    - Wrapper to respond to 0x02(read discret input) in the MODBUS specification. ////
////                                                                            ////
////-----------------------------------_>>  CODI 3  <<-----------------------------////
////  void modbus_read_holding_registers_rsp(address,byte_count,*reg_data)      ////
////    - Wrapper to respond to 0x03(read holding regs) in the MODBUS specification.  ////
////-----------------------------------_>>  CODI 3  <<-----------------------------////
////                                                                            ////
////  CODI 0x04                                                                 ////
////  void modbus_read_input_registers_rsp(address,byte_count,*input_data)      ////
////    - Wrapper to respond to 0x04(read input regs) in the MODBUS specification.    ////
////                                                                            ////
////  CODI 0x05                                                                 ////
////  void modbus_write_single_coil_rsp(address,output_address,output_value)    ////
////    - Wrapper to respond to 0x05(write single coil) in the MODBUS specification.  ////
////                                                                            ////
////-----------------------------------_>>  CODI 6  <<-----------------------------////
////  void modbus_write_single_register_rsp(address,reg_address,reg_value)      ////
////    - Wrapper to respond to 0x06(write single reg) in the MODBUS specification.   ////
////-----------------------------------_>>  CODI 6  <<-----------------------------////
```

```
////                                                                            ////
////  CODI 0x07                                                                 ////
////  void modbus_read_exception_status_rsp(address, data)                      ////
////    - Wrapper to respond to 0x07(read void status) in the MODBUS specification.   ////
////                                                                            ////
////  CODI 0x08                                                                 ////
////  void modbus_diagnostics_rsp(address,sub_func,data)                        ////
////    - Wrapper to respond to 0x08(diagnostics) in the MODBUS specification.       ////
////                                                                            ////
////  CODI 0x0B                                                                 ////
////  void modbus_get_comm_event_counter_rsp(address,status,event_count)        ////
////    - Wrapper to respond to 0x0B(get comm event count) in the MODBUS specification////
////                                                                            ////
////  CODI 0X0C                                                                 ////
////  void modbus_get_comm_event_log_rsp(address,status,event_count,message_count,    ////
////                                *events, events_len)                        ////
////    - Wrapper to respond to 0x0C(get comm event log) in the MODBUS specification. ////
////                                                                            ////
////  CODI 0X0F                                                                 ////
////  void modbus_write_multiple_coils_rsp(address,start_address,quantity)      ////
////    - Wrapper to respond to 0x0F(write multiple coils) in the MODBUS specification////
////                                                                            ////
////  CODI 0X10                                                                 ////
////  void modbus_write_multiple_registers_rsp(address,start_address,quantity)  ////
////    - Wrapper to respond to 0x10(write multiple regs) in the MODBUS specification.////
////                                                                            ////
////  CODI 0X11                                                                 ////
////  void modbus_report_slave_id_rsp(address,slave_id,run_status,*data,data_len)    ////
////    - Wrapper to respond to 0x11(report slave id) in the MODBUS specification.    ////
////                                                                            ////
////  CODI 0X14                                                                 ////
////  void modbus_read_file_record_rsp(address,byte_count,*request)             ////
////    - Wrapper to respond to 0x14(read file record) in the MODBUS specification.   ////
////                                                                            ////
////  CODI 0X15                                                                 ////
////  void modbus_write_file_record_rsp(address,byte_count,*request)            ////
////    - Wrapper to respond to 0x15(write file record) in the MODBUS specification.  ////
////                                                                            ////
////  CODI 0X16                                                                 ////
////  void modbus_mask_write_register_rsp(address,reference_address,AND_mask,OR_mask) ////
////    - Wrapper to respond to 0x16(read coils) in the MODBUS specification.         ////
////                                                                            ////
////  CODI 0X17                                                                 ////
////  void modbus_read_write_multiple_registers_rsp(address,*data,data_len)     ////
////    - Wrapper to respond to 0x17(read write mult regs) in the MODBUS specification////
////                                                                            ////
////  CODI 0X18                                                                 ////
////  void modbus_read_FIFO_queue_rsp(address,FIFO_len,*data)                   ////
////    - Wrapper to respond to 0x18(read FIFO queue) in the MODBUS specification.    ////
////                                                                            ////
////  void modbus_exception_rsp(int8 address, int16 func, exception error)      ////
////    - Wrapper to send an exception response.  See exception list below.     ////
////                                                                            ////
//// Exception List:                                                          ////
////  ILLEGAL_FUNCTION, ILLEGAL_DATA_ADDRESS, ILLEGAL_DATA_VALUE,               ////
////  SLAVE_DEVICE_FAILURE, ACKNOWLEDGE, SLAVE_DEVICE_BUSY, MEMORY_PARITY_ERROR,    ////
////  GATEWAY_PATH_UNAVAILABLE, GATEWAY_TARGET_NO_RESPONSE                      ////
////                                                                            ////
////////////////////////////////////////////////////////////////////////////////////
////                                                                            ////
////    Utilitzarem les funcions 3 (llegir) i 6 (escriure)                      ////
////                                                                            ////
////                                                                            ////
////////////////////////////////////////////////////////////////////////////////////



#use rs232(baud=MODBUS_SERIAL_BAUD, UART1, parity=N, stream=MODBUS_SERIAL)
#define RCV_OFF() {disable_interrupts(INT_RDA);}

#byte TXSTA=getenv("sfr:TXSTA")
#bit TRMT=TXSTA.1

// "INTERUPCIÓ" DE FINAL DE TRANSMISSIÓ, FETA PER POOLING (DELAY)
#define WAIT_FOR_HW_BUFFER()\
{\
   while(!TRMT);\
}

//Arriba nova trama a tractar.
int1 modbus_serial_new=0;
```

```
////////////////////////////////////////////////////////////////////////////////////
//**********************************************************************************//
//Les següents excepcions estan definides en el protocol de MODBUS.                 //
//Son perquè l'esclau pogui comunicar  els problemes de comunicació                 //
//amb el mestre. La excepció de Timeout, es retorna quan no hi ha cap               //
//esclau que respongui al master durant aquest temps.                               //
//                                                                                  //
// TIPEDEF  DEFINEIX UN TIPUS. DEFINEIX CADENES DE TEXT                             //
// CADA CADENA DE TXT QUEDA ASSOCIADA A UN NUMERO                                   //
//**********************************************************************************//
////////////////////////////////////////////////////////////////////////////////////

typedef enum _exception{ILLEGAL_FUNCTION=1,ILLEGAL_DATA_ADDRESS=2,
ILLEGAL_DATA_VALUE=3,SLAVE_DEVICE_FAILURE=4,ACKNOWLEDGE=5,SLAVE_DEVICE_BUSY=6,
MEMORY_PARITY_ERROR=8,GATEWAY_PATH_UNAVAILABLE=10,GATEWAY_TARGET_NO_RESPONSE=11,
TIMEOUT=12} exception;


////////////////////////////////////////////////////////////////////////////////////
//**********************************************************************************//
// Aquestes funcions estan definides en el protocol de MODBUS.  Poden ser utilitzades //
// pel esclau per comprobar la funcio d entrada.                                    //
//**********************************************************************************//
////////////////////////////////////////////////////////////////////////////////////

typedef enum _function{FUNC_READ_COILS=0x01,FUNC_READ_DISCRETE_INPUT=0x02,
FUNC_READ_HOLDING_REGISTERS=0x03,FUNC_READ_INPUT_REGISTERS=0x04,
FUNC_WRITE_SINGLE_COIL=0x05,FUNC_WRITE_SINGLE_REGISTER=0x06,
FUNC_READ_EXCEPTION_STATUS=0x07,FUNC_DIAGNOSTICS=0x08,
FUNC_GET_COMM_EVENT_COUNTER=0x0B,FUNC_GET_COMM_EVENT_LOG=0x0C,
FUNC_WRITE_MULTIPLE_COILS=0x0F,FUNC_WRITE_MULTIPLE_REGISTERS=0x10,
FUNC_REPORT_SLAVE_ID=0x11,FUNC_READ_FILE_RECORD=0x14,
FUNC_WRITE_FILE_RECORD=0x15,FUNC_MASK_WRITE_REGISTER=0x16,
FUNC_READ_WRITE_MULTIPLE_REGISTERS=0x17,FUNC_READ_FIFO_QUEUE=0x18} function;


////////////////////////////////////////////////////////////////////////////////////
//**********************************************************************************//
// Estats de la recepció de trames MODBUS.                                          //
// 1r Adreçaement                                                                   //
// 2n Codi de funció                                                                //
// 3r Dades                                                                         //
//**********************************************************************************//
////////////////////////////////////////////////////////////////////////////////////
enum {MODBUS_GETADDY=0, MODBUS_GETFUNC=1, MODBUS_GETDATA=2} modbus_serial_state = 0;

// La instrucció Union fa que d estigui format per 2 bytes b1 i b0

union
{
   int8 b[2];  // queden definits 2 bytes B0 i B1
   int16 d;    // Defineix un word de 16 bits
} modbus_serial_crc;


////////////////////////////////////////////////////////////////////////////////////
//**********************************************************************************//
// Aquest struct és un conjunt de bytes que modifiquem en la Recepció.              //
//                                                                                  //
// La funció struct ens serveix per tenir variables ordenades per grups. "Carpeta.byte" //
// exemple: modbus_rx.adress // Aquestes bytes queden modificats en rebere la trama     //
//**********************************************************************************//
////////////////////////////////////////////////////////////////////////////////////

struct
{
   int8 address;
   int8 len;                              //numero de bytes del missatge rebut
   function func;                         //codifuncio del missatge rebut
   exception error;                       //error si n'hi ha
   int8 data[MODBUS_SERIAL_RX_BUFFER_SIZE]; //dades del missatge rebut
} modbus_rx;
```

```
/* Taula de càlcul del CRC_high-order byte*/
const unsigned char modbus_auchCRCHi[] = {
0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,
0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,
0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,
0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,
0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,
0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,
0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,
0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,
0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,
0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,
0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,
0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,
0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,
0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,
0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,
0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,
0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,
0x40
};



/* Taula de càlcul del CRC_low-order byte*/
const char modbus_auchCRCLo[] = {
0x00,0xC0,0xC1,0x01,0xC3,0x03,0x02,0xC2,0xC6,0x06,0x07,0xC7,0x05,0xC5,0xC4,
0x04,0xCC,0x0C,0x0D,0xCD,0x0F,0xCF,0xCE,0x0E,0x0A,0xCA,0xCB,0x0B,0xC9,0x09,
0x08,0xC8,0xD8,0x18,0x19,0xD9,0x1B,0xDB,0xDA,0x1A,0x1E,0xDE,0xDF,0x1F,0xDD,
0x1D,0x1C,0xDC,0x14,0xD4,0xD5,0x15,0xD7,0x17,0x16,0xD6,0xD2,0x12,0x13,0xD3,
0x11,0xD1,0xD0,0x10,0xF0,0x30,0x31,0xF1,0x33,0xF3,0xF2,0x32,0x36,0xF6,0xF7,
0x37,0xF5,0x35,0x34,0xF4,0x3C,0xFC,0xFD,0x3D,0xFF,0x3F,0x3E,0xFE,0xFA,0x3A,
0x3B,0xFB,0x39,0xF9,0xF8,0x38,0x28,0xE8,0xE9,0x29,0xEB,0x2B,0x2A,0xEA,0xEE,
0x2E,0x2F,0xEF,0x2D,0xED,0xEC,0x2C,0xE4,0x24,0x25,0xE5,0x27,0xE7,0xE6,0x26,
0x22,0xE2,0xE3,0x23,0xE1,0x21,0x20,0xE0,0xA0,0x60,0x61,0xA1,0x63,0xA3,0xA2,
0x62,0x66,0xA6,0xA7,0x67,0xA5,0x65,0x64,0xA4,0x6C,0xAC,0xAD,0x6D,0xAF,0x6F,
0x6E,0xAE,0xAA,0x6A,0x6B,0xAB,0x69,0xA9,0xA8,0x68,0x78,0xB8,0xB9,0x79,0xBB,
0x7B,0x7A,0xBA,0xBE,0x7E,0x7F,0xBF,0x7D,0xBD,0xBC,0x7C,0xB4,0x74,0x75,0xB5,
0x77,0xB7,0xB6,0x76,0x72,0xB2,0xB3,0x73,0xB1,0x71,0x70,0xB0,0x50,0x90,0x91,
0x51,0x93,0x53,0x52,0x92,0x96,0x56,0x57,0x97,0x55,0x95,0x94,0x54,0x9C,0x5C,
0x5D,0x9D,0x5F,0x9F,0x9E,0x5E,0x5A,0x9A,0x9B,0x5B,0x99,0x59,0x58,0x98,0x88,
0x48,0x49,0x89,0x4B,0x8B,0x8A,0x4A,0x4E,0x8E,0x8F,0x4F,0x8D,0x4D,0x4C,0x8C,
0x44,0x84,0x85,0x45,0x87,0x47,0x46,0x86,0x82,0x42,0x43,0x83,0x41,0x81,0x80,
0x40
};



////----------------------------------->>  RCV_ON  <<-----------------------------------////
//////////////////////////////////////////////////////////////////////////////////////////
//*****************************************************************************************//
// Objectiu:    Permet recepció de dades                                                   //
// Entrades:    Cap                                                                        //
// Sortides:    Cap                                                                        //
//*****************************************************************************************//
//////////////////////////////////////////////////////////////////////////////////////////

void RCV_ON(void)
{
        while(kbhit(MODBUS_SERIAL))
        {   getc();                     // Borra Buffer de la Usart.
        }
        clear_interrupt(INT_RDA);       // Neteja interrupció RDA
        enable_interrupts(INT_RDA);     // Habilita interrupció RDA

        // la interrupció RDA es dispara quan es té el buffer serie ple
}
```

```
////////////////////////////////////////////////////////////////////////////////////////
//************************************************************************************//
//**********************************                  ********************************//
//**********************************  CONTROL DEL 485  ********************************//
//**********************************                  ********************************//
//************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////


////----------------------------->>  MODBUS_INIT  <<-----------------------------////
////////////////////////////////////////////////////////////////////////////////////////
//************************************************************************************//
// Objectiu: Inicialitza comunicació Rs485.                                           //
// Entrades: Cap                                                                      //
// Sortides: Cap                                                                      //
//************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////

void modbus_init()
{
    output_low(PIN_C5);                // Configura el MAX mode lectura
    delay_us(200);                     // Retard per esperar config.Max mode lectura
    RCV_ON();                          // Crida aquesta funció per activar la recepció
    setup_timer_2(T2_DIV_BY_16,230,3);  // Configuració del Timer 2 per interrupció ~4ms.

    // Prescale 16, Periode 230, Postcales 3, cristall 11Mhz
        // clock de 11592000 (11MHz)
        // (1/clock)*4*t2_div*(periode+1)= 1/11592000*4*16*231= 1,2753ms
        // Per tenir interrupció voldrem 3 rsts -> 1,2753ms*3 = 3,8 ms ~ 4ms de timeout

    enable_interrupts(GLOBAL);
    // permet les interrupcions T1(int_rapida),T2 (timeout)i RDA (comunicació)
}


////-------------------------->>  MODBUS_ENABLE_TIMEOUT  <<--------------------------////
////////////////////////////////////////////////////////////////////////////////////////
//************************************************************************************//
// Objectiu:    Inicia el timer de Timeout alhora d'enviar o rebre trames.            //
// Entrades:    Enable, s'utilitza per canviar l'estat on/off                         //
// Sortides:    cap                                                                   //
// El timeout s'haurà d'activar quan el següent byte tardi més de 4ms                 //
// a 1 = Carrega timeout                                                              //
// a 0 = Parat                                                                        //
// Per enviar trama activem el Timeout                                                //
//************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////

void modbus_enable_timeout(int1 enable)
{
   disable_interrupts(INT_TIMER2);      // Desabilita la interrupció del timer 2
   if (enable) {
      set_timer2(0);                    // posa a 0 el timer
      clear_interrupt(INT_TIMER2);      // neteja interrupció
      enable_interrupts(INT_TIMER2);    // Torna a habilitar la int del timer
   }
}


////----------------------------->>  MODBUS_TIMEOUT_NOW  <<-----------------------------////
////////////////////////////////////////////////////////////////////////////////////////
//************************************************************************************//
// Objectiu:    Quan s'ha disparat la interrupció 2 el programa es situa a int_timer2 //
// Entrades:    Cap                                                                   //
// Sortides:    Cap                                                                   //
//************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////

#int_timer2                         //
void modbus_timeout_now(void)       // EL MODBUS-GETDATA és l'estat 3 definit a enum.

// Si el programa es troba en l'Estat 2 i el Crc ha arribat a 0 (trama correcta) i tenim trama nova:
{
   if((modbus_serial_state == MODBUS_GETDATA) && (modbus_serial_crc.d == 0x0000) &&
(!modbus_serial_new))
   {
      modbus_rx.len-=2;             // Resta 2 bytes en el buffer per
                                    // per no comptar els bytes del CRC en la llargada
      modbus_serial_new=TRUE;       // Marcador que indica que hi ha una nova trama a processar
   }
   else
```

```
      modbus_serial_new=FALSE;

    // prepara micro per la següent trama.
   modbus_serial_crc.d=0xFFFF;         // Inicialitza el CRC (sempre inicialitzat a 0xFFFF)
   modbus_serial_state=MODBUS_GETADDY; // Col·loca la comunicació en E0 de llegir adreçaments.
   modbus_enable_timeout(FALSE);       // Para temporitzador de timeout (Enable = False = 0).
}


////----------------------------->>  MODBUS_CALC_CRC  <<-----------------------------////
////////////////////////////////////////////////////////////////////////////////////////
//****************************************************************************************//
// Objectiu:    Calcula el CRC de l'ultim byte que se li passa per la variable Data     //
//              (Data = C de Modbus_serial_putc) i actualitza el CRC glogal             //
// Entrades:    Caràcter                                                                //
// Sortides:    Cap                                                                     //
//****************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////

void modbus_calc_crc(char data)
{
  int8 uIndex ; // És l'index per adreçar-nos a les taules de CRC (lookup table)

  // Càlcul del CRC

  uIndex = (modbus_serial_crc.b[1]) ^ data;
  modbus_serial_crc.b[1] = (modbus_serial_crc.b[0]) ^ modbus_auchCRCHi[uIndex]; //byte major pes
  modbus_serial_crc.b[0] = modbus_auchCRCLo[uIndex];                            //byte menor pes
}


////----------------------------->>  MODBUS_SERIAL_PUTC  <<-----------------------------////
////////////////////////////////////////////////////////////////////////////////////////
//****************************************************************************************//
// Objectiu:    Posa un byte a la Uart per poder enviar trames                          //
// Entrades:    Caràcter                                                                //
// Sortides:    Cap                                                                     //
//****************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////

void modbus_serial_putc(int8 c)
{
   fputc(c, MODBUS_SERIAL);              // Enviem el byte C a la Uart.
   modbus_calc_crc(c);                   // Enviem el byte C a la funció per calcular CRC (data).
   delay_us(1000000/MODBUS_SERIAL_BAUD); // Retard perquè receptor tingui temps de calcular CRC.
}


////--------------------------->>  INCOMING_MODBUS_SERIAL  <<--------------------------////
////////////////////////////////////////////////////////////////////////////////////////
//****************************************************************************************//
// Objectiu:   És la interrupció RDA que s'activa quan tenim buffer ple                 //
// Entrades:   Cap                                                                      //
// Sortides:   Cap                                                                      //
//****************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////

#int_rda                            // Quan tenim buffer serie ple es dispara una interrupció.
void incomming_modbus_serial() {
   char c;

   c=fgetc(MODBUS_SERIAL);      // MODBUS SERIAL és el nom que hem donat a la USART.
                                // Agafa sempre tota la trama encara que no sigui per nosaltres.
   if (!modbus_serial_new)      // Mira en quin estat ens trobem ADR, FUN, DAD.
   {
      if(modbus_serial_state == MODBUS_GETADDY) // E0.
      {
         modbus_serial_crc.d = 0xFFFF; // Inicialitzar CRC per rebre.
         modbus_rx.address = c;        // Guardar adreça que arriba.
         modbus_serial_state++;        // Incrementa Estat.
         modbus_rx.len = 0;            // Inicia byte llargada.
         modbus_rx.error=0;            // Inicia byte d'error.
      }
      else if(modbus_serial_state == MODBUS_GETFUNC)  // E1.
      {
         modbus_rx.func = c;           // guarda Codi de funció.
         modbus_serial_state++;        // Incrementa l'estat del sistema.
      }
      else if(modbus_serial_state == MODBUS_GETDATA)  // E2.
      // el rx buffer size és el màxim de memòria que li haguem donat (definit a prg Main).
      {
         if (modbus_rx.len>=MODBUS_SERIAL_RX_BUFFER_SIZE)
```

```
{modbus_rx.len=MODBUS_SERIAL_RX_BUFFER_SIZE-1;}
         modbus_rx.data[modbus_rx.len]=c;   // Guarda la C a rx.data
         modbus_rx.len++;                   // Incrementa la posició
      }
      modbus_calc_crc(c);              // Calcula el CRC del valor que agafem de la Uart.
      modbus_enable_timeout(TRUE);     // Activem el temps de timeout i el següent byte
                                       // té 4 ms de timeout per arribar.
   }
}


////-------------------------->>  MODBUS_SERIAL_SEND_START  <<-------------------------////
//////////////////////////////////////////////////////////////////////////////////////////
//*****************************************************************************************//
// Objectiu:   Enviar missatge pel bus                                                     //
// Entrades:   1) L'adreçament del receptor (to)                                           //
//             2) Nombre de bytes de dades a enviar                                        //
//             3) Un punter per les dades a enviar                                         //
//             4) La llargada de les dades                                                 //
// Sortides:   TRUE                                                                        //
//             FALSE                                                                       //
// Note:       Format:  source | destination | data-length | data | checksum             //
//*****************************************************************************************//
//////////////////////////////////////////////////////////////////////////////////////////

void modbus_serial_send_start(int8 to, int8 func)
{
    modbus_serial_crc.d=0xFFFF;             // Inicialitza el CRC.
    modbus_serial_new=FALSE;                // re-inicialitzem el bit de nova trama.

    RCV_OFF();                              // Parem la Recepció (off).

    output_high(PIN_C5);                    // Prepara el Max per Transmetre.
    delay_us(200);                          // Retard de 100 per config.Max Transmetre.

    delay_us(3500000/MODBUS_SERIAL_BAUD);   // 3.5 character delay

    modbus_serial_putc(to);                 // envia les dades a la uart.
    modbus_serial_putc(func);               // envia les dades a la uart.
                                            // de les anteriors dades també s'en calcularà
                                            // el checksum pq es crida a Modbus_calc_crc.
}

////-------------------------->>  MODBUS_SERIAL_SEND_STOP  <<-------------------------////
//////////////////////////////////////////////////////////////////////////////////////////
//*****************************************************************************************//
//                                                                                         //
//                                                                                         //
//*****************************************************************************************//
//////////////////////////////////////////////////////////////////////////////////////////
void modbus_serial_send_stop()
{
    int8 crc_low, crc_high;

    crc_high=modbus_serial_crc.b[1];    // Guardem el valor del checksum MSB.
    crc_low=modbus_serial_crc.b[0];     // Guardem el valor del checksum LSB.

    // Per defecte es calcularà el Crc d'aquests dos valors però enviarem a
    // la Uart directamentel valor crc high i low.
    modbus_serial_putc(crc_high);
    modbus_serial_putc(crc_low);        // Enviem aquests valors a la Uart.

    WAIT_FOR_HW_BUFFER();

    delay_us(3500000/MODBUS_SERIAL_BAUD); //3.5 character delay.

    RCV_ON();                           // Activa recepció.
    output_low(PIN_C5);                 // Max mode Recepció
    delay_us(200);                      // Retard de 100 per config.Max Recepció.

    modbus_serial_crc.d=0xFFFF;
}
```

```
////------------------------------>>  MODBUS_KBHIT  <<------------------------------////
////////////////////////////////////////////////////////////////////////////////////////
//****************************************************************************************//
// Objectiu:   Guarda missate del bus en un buffer.                                       //
// Entrades:   Cap                                                                        //
// Sortides:   TRUE si el missatge s'ha rebut                                             //
//             FALSE                                                                      //
//****************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////////

int1 modbus_kbhit()
{
   if(!modbus_serial_new)                  // Si no tenim nova trama Kbhit=0.
      return FALSE;
   else if(modbus_rx.func & 0x80)          // Sino si funció te un error (0x8?).
   {
      modbus_rx.error = modbus_rx.data[0];  // Guarda l'error.
      modbus_rx.len = 1;                   // es modifica la trama a enviar.
   }
   modbus_serial_new=FALSE;                // Inicia l'indicador de trama nova.
   return TRUE;                            // Kbhit=1 TENIM NOVA TRAMA
}



////////////////////////////////////////////////////////////////////////////////////////////
//****************************************************************************************//
//****************************                                    ************************//
//****************************  Funcions de l'esclau Modbus  ************************//
//****************************                                    ************************//
//****************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////
//****************************************************************************************//
// Els seguents structs, s'utilitzen per  fer peticions o respostes de lectura o        //
// escriptura.                                                                            //
//****************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////////

typedef struct _modbus_read_sub_request_rsp
{
   int8 record_length;
   int8 reference_type;
   int16 data[((MODBUS_SERIAL_RX_BUFFER_SIZE)/2)-3];
} modbus_read_sub_request_rsp;

typedef struct _modbus_write_sub_request_rsp
{
   int8 reference_type;
   int16 file_number;
   int16 record_number;
   int16 record_length;
   int16 data[((MODBUS_SERIAL_RX_BUFFER_SIZE)/2)-8];
} modbus_write_sub_request_rsp;

////////////////////////////////////////////////////////////////////////////////////////////
//****************************************************************************************//
// Les següents funcions de l'esclau, estan definides en el protocol de Modbus.         //
// Aquest protocol es pot trobar a la web http://www.modbus.org                           //
// Totes les funcions envien primer l'adreçament de l'esclau a qui va dirigida la funcio//
//****************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////////

////---------------------------------->>  CODI 1  <<----------------------------------////
/*
read_coils_rsp
Entrada:     int8       address            Slave Address
           int8       byte_count         Number of bytes being sent
           int8*      coil_data          Pointer to an array of data to send
Sortida:    void
*/
void modbus_read_coils_rsp(int8 address, int8 byte_count, int8* coil_data)
{
   int8 i;
   modbus_serial_send_start(address, FUNC_READ_COILS);

   modbus_serial_putc(byte_count);

   for(i=0; i < byte_count; ++i)
   {
```

```
      modbus_serial_putc(*coil_data);
      coil_data++;
   }

   modbus_serial_send_stop();
}


////---------------------------------->>  CODI 2  <<----------------------------------////
/*
read_discrete_input_rsp
Entrada:     int8       address            Slave Address
           int8       byte_count         Number of bytes being sent
           int8*      input_data         Pointer to an array of data to send
Sortida:    void
*/
void modbus_read_discrete_input_rsp(int8 address, int8 byte_count,
                                    int8 *input_data)
{
   int8 i;
   modbus_serial_send_start(address, FUNC_READ_DISCRETE_INPUT);

   modbus_serial_putc(byte_count);

   for(i=0; i < byte_count; ++i)
   {
      modbus_serial_putc(*input_data);
      input_data++;
   }

   modbus_serial_send_stop();
}


////---------------------------------->>  CODI 3  <<----------------------------------////
////////////////////////////////////////////////////////////////////////////////////////
//*****************************************************************************************//
// read_holding_registers_rsp       // Master vol llegir de l'eslau                       //
// Entrada:   int8       address            Adreça de l'esclau                           //
//            int8       byte_count         Nombre de bytes que s'envien                 //
//            int8*      reg_data           Punter en un vector de dades a enviar        //
//                                          Punter de READ HOLDING REGISTERS             //
// Sortida:    void                                                                      //
//*****************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////////
void modbus_read_holding_registers_rsp(int8 address, int8 byte_count,
                                       int8 *reg_data)
{
   int8 i;
   modbus_serial_send_start(address, FUNC_READ_HOLDING_REGISTERS); // Send Start

   modbus_serial_putc(byte_count);

   for(i=0; i < byte_count; ++i)
   {
      modbus_serial_putc(*reg_data);  // amb * es refereix al contingut
      reg_data++;                     // sense * a la posició// Incrementem posició
   }

   modbus_serial_send_stop();         // Send Stop
}

////---------------------------------->>  CODI 4  <<----------------------------------////
/*
read_input_registers_rsp
Entrada:     int8       address            Slave Address
           int8       byte_count         Number of bytes being sent
           int8*      input_data         Pointer to an array of data to send
Sortida:    void
*/
void modbus_read_input_registers_rsp(int8 address, int8 byte_count,
                                       int8 *input_data)
{
   int8 i;

   modbus_serial_send_start(address, FUNC_READ_INPUT_REGISTERS);

   modbus_serial_putc(byte_count);

   for(i=0; i < byte_count; ++i)
   {
      modbus_serial_putc(*input_data);
```

```
      input_data++;
   }

   modbus_serial_send_stop();
}

////---------------------------------->>  CODI 5  <<----------------------------------////
/*
write_single_coil_rsp
Entrada:     int8       address            Slave Address
           int16      output_address     Echo of output address received
           int16      output_value       Echo of output value received
Sortida:    void
*/
void modbus_write_single_coil_rsp(int8 address, int16 output_address, int16 output_value)
{
   modbus_serial_send_start(address, FUNC_WRITE_SINGLE_COIL);

   modbus_serial_putc(make8(output_address,1));
   modbus_serial_putc(make8(output_address,0));

   modbus_serial_putc(make8(output_value,1));
   modbus_serial_putc(make8(output_value,0));

   modbus_serial_send_stop();
}


////---------------------------------->>  CODI 6  <<----------------------------------////
////////////////////////////////////////////////////////////////////////////////////////
//*****************************************************************************************//
// write_single_register_rsp            // Escriure un sol Registre                        //
// Entrada:   int8       address            Adreçament de l'esclau                        //
//            int16      reg_address        Echo of register address received             //
//            int16      reg_value          Echo of register value received               //
// Sortida:   void                                                                         //
//*****************************************************************************************//
////////////////////////////////////////////////////////////////////////////////////////////
void modbus_write_single_register_rsp(int8 address, int16 reg_address, int16 reg_value)
{

   modbus_serial_send_start(address, FUNC_WRITE_SINGLE_REGISTER); // Send Start

   modbus_serial_putc(make8(reg_address,1));     // Posició d'inici escriptura
   modbus_serial_putc(make8(reg_address,0));

   modbus_serial_putc(make8(reg_value,1));       // nº de registres
   modbus_serial_putc(make8(reg_value,0));

   modbus_serial_send_stop();                   // Send Stop
}

////---------------------------------->>  CODI 7  <<----------------------------------////
/*
read_exception_status_rsp
Entrada:     int8       address            Slave Address
Sortida:    void
*/
void modbus_read_exception_status_rsp(int8 address, int8 data)
{
   modbus_serial_send_start(address, FUNC_READ_EXCEPTION_STATUS);
   modbus_serial_send_stop();
}

////---------------------------------->>  CODI 8  <<----------------------------------////
/*
diagnostics_rsp
Entrada:     int8       address            Slave Address
           int16      sub_func           Echo of sub function received
           int16      data               Echo of data received
Sortida:    void
*/
void modbus_diagnostics_rsp(int8 address, int16 sub_func, int16 data)
{
   modbus_serial_send_start(address, FUNC_DIAGNOSTICS);

   modbus_serial_putc(make8(sub_func,1));
   modbus_serial_putc(make8(sub_func,0));

   modbus_serial_putc(make8(data,1));
   modbus_serial_putc(make8(data,0));
```

```
   modbus_serial_send_stop();
}

////---------------------------------->>  CODI 0B  <<----------------------------------////
/*
get_comm_event_counter_rsp
Entrada:     int8       address            Slave Address
           int16      status             Status, refer to MODBUS documentation
           int16      event_count        Count of events
Sortida:    void
*/
void modbus_get_comm_event_counter_rsp(int8 address, int16 status,
                                       int16 event_count)
{
   modbus_serial_send_start(address, FUNC_GET_COMM_EVENT_COUNTER);

   modbus_serial_putc(make8(status, 1));
   modbus_serial_putc(make8(status, 0));

   modbus_serial_putc(make8(event_count, 1));
   modbus_serial_putc(make8(event_count, 0));

   modbus_serial_send_stop();
}

////---------------------------------->>  CODI 0C  <<----------------------------------////
/*
get_comm_event_counter_rsp
Entrada:     int8       address            Slave Address
           int16      status             Status, refer to MODBUS documentation
           int16      event_count        Count of events
           int16      message_count      Count of messages
           int8*      events             Pointer to event data
           int8       events_len         Length of event data in bytes
Sortida:    void
*/
void modbus_get_comm_event_log_rsp(int8 address, int16 status,
                                   int16 event_count, int16 message_count,
                                   int8 *events, int8 events_len)
{
   int8 i;

   modbus_serial_send_start(address, FUNC_GET_COMM_EVENT_LOG);

   modbus_serial_putc(events_len+6);

   modbus_serial_putc(make8(status, 1));
   modbus_serial_putc(make8(status, 0));

   modbus_serial_putc(make8(event_count, 1));
   modbus_serial_putc(make8(event_count, 0));

   modbus_serial_putc(make8(message_count, 1));
   modbus_serial_putc(make8(message_count, 0));

   for(i=0; i < events_len; ++i)
   {
      modbus_serial_putc(*events);
      events++;
   }

   modbus_serial_send_stop();
}

////---------------------------------->>  CODI 0F  <<----------------------------------////
/*
write_multiple_coils_rsp
Entrada:     int8       address            Slave Address
           int16      start_address      Echo of address to start at
           int16      quantity           Echo of amount of coils written to
Sortida:    void
*/
void modbus_write_multiple_coils_rsp(int8 address, int16 start_address,
                                     int16 quantity)
{
   modbus_serial_send_start(address, FUNC_WRITE_MULTIPLE_COILS);

   modbus_serial_putc(make8(start_address,1));
   modbus_serial_putc(make8(start_address,0));

   modbus_serial_putc(make8(quantity,1));
   modbus_serial_putc(make8(quantity,0));
```

```
   modbus_serial_send_stop();
}

////---------------------------------->>  CODI 10  <<----------------------------------////
/*
write_multiple_registers_rsp

Entrada:     int8       address            Slave Address
           int16      start_address      Echo of address to start at
           int16      quantity           Echo of amount of registers written to
Sortida:    void
*/
void modbus_write_multiple_registers_rsp(int8 address, int16 start_address,
                                         int16 quantity)
{
   modbus_serial_send_start(address, FUNC_WRITE_MULTIPLE_REGISTERS);

   modbus_serial_putc(make8(start_address,1));
   modbus_serial_putc(make8(start_address,0));

   modbus_serial_putc(make8(quantity,1));
   modbus_serial_putc(make8(quantity,0));

   modbus_serial_send_stop();
}

////---------------------------------->>  CODI 11  <<----------------------------------////
/*
report_slave_id_rsp
Entrada:     int8       address            Slave Address
           int8       slave_id           Slave Address
           int8       run_status         Are we running?
           int8*      data               Pointer to an array holding the data
           int8       data_len           Length of data in bytes
Sortida:    void
*/
void modbus_report_slave_id_rsp(int8 address, int8 slave_id, int1 run_status, int8 *data, int8 data_len)
{
   int8 i;

   modbus_serial_send_start(address, FUNC_REPORT_SLAVE_ID);

   modbus_serial_putc(data_len+2);
   modbus_serial_putc(slave_id);

   if(run_status)
    modbus_serial_putc(0xFF);
   else
    modbus_serial_putc(0x00);

   for(i=0; i < data_len; ++i)
   {
      modbus_serial_putc(*data);
      data++;
   }

   modbus_serial_send_stop();
}

////---------------------------------->>  CODI 14  <<----------------------------------////
/*
read_file_record_rsp
Entrada:     int8                    address            Slave Address
           int8                    byte_count         Number of bytes to send
           read_sub_request_rsp*   request            Structure holding record/data information
Sortida:    void
*/
void modbus_read_file_record_rsp(int8 address, int8 byte_count,
                                 modbus_read_sub_request_rsp *request)
{
   int8 i=0,j;

   modbus_serial_send_start(address, FUNC_READ_FILE_RECORD);

   modbus_serial_putc(byte_count);

   while(i < byte_count);
   {
      modbus_serial_putc(request->record_length);
      modbus_serial_putc(request->reference_type);
```

```
      for(j=0; (j < request->record_length); j+=2)
      {
         modbus_serial_putc(make8(request->data[j], 1));
         modbus_serial_putc(make8(request->data[j], 0));
      }

      i += (request->record_length)+1;
      request++;
   }

   modbus_serial_send_stop();
}

////--------------------------------->>  CODI 15  <<---------------------------------////
/*
write_file_record_rsp
Entrada:     int8                    address           Slave Address
           int8                    byte_count        Echo of number of bytes sent
           write_sub_request_rsp*  request           Echo of Structure holding record information
Sortida:    void
*/
void modbus_write_file_record_rsp(int8 address, int8 byte_count,
                                  modbus_write_sub_request_rsp *request)
{
   int8 i, j=0;

   modbus_serial_send_start(address, FUNC_WRITE_FILE_RECORD);

   modbus_serial_putc(byte_count);

   for(i=0; i < byte_count; i+=(7+(j*2)))
   {
      modbus_serial_putc(request->reference_type);
      modbus_serial_putc(make8(request->file_number, 1));
      modbus_serial_putc(make8(request->file_number, 0));
      modbus_serial_putc(make8(request->record_number, 1));
      modbus_serial_putc(make8(request->record_number, 0));
      modbus_serial_putc(make8(request->record_length, 1));
      modbus_serial_putc(make8(request->record_length, 0));

      for(j=0; (j < request->record_length); j+=2)
      {
         modbus_serial_putc(make8(request->data[j], 1));
         modbus_serial_putc(make8(request->data[j], 0));
      }
   }

   modbus_serial_send_stop();
}
////--------------------------------->>  CODI 16  <<---------------------------------////
/*
mask_write_register_rsp
Entrada:     int8        address           Slave Address
           int16       reference_address  Echo of reference address
           int16       AND_mask           Echo of AND mask
           int16       OR_mask            Echo or OR mask
Sortida:    void
*/
void modbus_mask_write_register_rsp(int8 address, int16 reference_address,
                          int16 AND_mask, int16 OR_mask)
{
   modbus_serial_send_start(address, FUNC_MASK_WRITE_REGISTER);

   modbus_serial_putc(make8(reference_address,1));
   modbus_serial_putc(make8(reference_address,0));

   modbus_serial_putc(make8(AND_mask,1));
   modbus_serial_putc(make8(AND_mask,0));

   modbus_serial_putc(make8(OR_mask, 1));
   modbus_serial_putc(make8(OR_mask, 0));

   modbus_serial_send_stop();
}
```

```
////---------------------------------->>  CODI 17  <<----------------------------------////
/*
read_write_multiple_registers_rsp
Entrada:     int8        address            Slave Address
           int16*      data               Pointer to an array of data
           int8        data_len           Length of data in bytes
Sortida:    void
*/
void modbus_read_write_multiple_registers_rsp(int8 address, int8 data_len,
                                              int16 *data)
{
   int8 i;

   modbus_serial_send_start(address, FUNC_READ_WRITE_MULTIPLE_REGISTERS);

   modbus_serial_putc(data_len*2);

   for(i=0; i < data_len*2; i+=2)
   {
      modbus_serial_putc(make8(data[i], 1));
      modbus_serial_putc(make8(data[i], 0));
   }

   modbus_serial_send_stop();
}

////---------------------------------->>  CODI 18  <<----------------------------------////
/*
read_FIFO_queue_rsp
Entrada:     int8        address
           int16       FIFO_len
           int16*      data
Sortida:    void
*/
void modbus_read_FIFO_queue_rsp(int8 address, int16 FIFO_len, int16 *data)
{
   int8 i;
   int16 byte_count;

   byte_count = ((FIFO_len*2)+2);

   modbus_serial_send_start(address, FUNC_READ_FIFO_QUEUE);

   modbus_serial_putc(make8(byte_count, 1));
   modbus_serial_putc(make8(byte_count, 0));

   modbus_serial_putc(make8(FIFO_len, 1));
   modbus_serial_putc(make8(FIFO_len, 0));

   for(i=0; i < FIFO_len; i+=2)
   {
      modbus_serial_putc(make8(data[i], 1));
      modbus_serial_putc(make8(data[i], 0));
   }

   modbus_serial_send_stop();
}


////--------------------------------->>  Excepcions  <<----------------------------------////


void modbus_exception_rsp(int8 address, int16 func, exception error)
{
   modbus_serial_send_start(address, func|0x80);
   modbus_serial_putc(error);
   modbus_serial_send_stop();
}

int8 swap_bits(int8 c)
{
   return ((c&1)?128:0)|((c&2)?64:0)|((c&4)?32:0)|((c&8)?16:0)|((c&16)?8:0)
          |((c&32)?4:0)|((c&64)?2:0)|((c&128)?1:0);
}
```

```
////////////////////////////////////////////////////////////////////////////////////
//*********************************************************************************//
//*********************************                   *******************************//
//*********************************  Interpret Modbus  *******************************//
//*********************************                   *******************************//
//*********************************************************************************//
////////////////////////////////////////////////////////////////////////////////////

void interpret_modbus (void)
{
    if ((modbus_rx.address == mod_adr) || modbus_rx.address == 0) //Si no és l'adreça del nostre esclau
ja no cal fer res+
    {
        led_14_on
        delay_ms(100);
        switch(modbus_rx.func)
        {
            case FUNC_READ_COILS:
            case FUNC_READ_DISCRETE_INPUT:
                if(modbus_rx.data[0] || modbus_rx.data[2] || modbus_rx.data[1] >= 8 ||
modbus_rx.data[3]+modbus_rx.data[1] > 8)
                    modbus_exception_rsp(mod_adr,modbus_rx.func,ILLEGAL_DATA_ADDRESS);
                else
                {
                    int8 data;
                    if(modbus_rx.func == FUNC_READ_COILS)
                        data = coils>>(modbus_rx.data[1]);
                    else
                        data = Inputs>>(modbus_rx.data[1]);

                    data = data & (0xFF>>(8-modbus_rx.data[3]));
                    if(modbus_rx.func == FUNC_READ_COILS)
                        modbus_read_discrete_input_rsp(mod_adr, 0x01, &data);
                    else
                        modbus_read_discrete_input_rsp(mod_adr, 0x01, &data);

                    event_count++;
                }
                break;

//************************************   Funcio 3  *********************************//
                case FUNC_READ_HOLDING_REGISTERS:           // Funció 3
            case FUNC_READ_INPUT_REGISTERS:             // Funció 4

                if(modbus_rx.data[0] || modbus_rx.data[2] ||modbus_rx.data[1] >= 8 ||
modbus_rx.data[3]+modbus_rx.data[1] > 8)
                // mira que la suma no passi de 8 pq ja seria més gran que el registre holdregs
                {
                    modbus_exception_rsp(mod_adr,modbus_rx.func,ILLEGAL_DATA_ADDRESS);
                    // s'hauria de enviar una resposta d'error  // Error mapa de memòria
                   }
                else
                {
                    if(modbus_rx.func == FUNC_READ_HOLDING_REGISTERS)
                    // Comprova si es vol entrar a Read holding registers o a la F 0x04



modbus_read_holding_registers_rsp(mod_adr,(modbus_rx.data[3]*2),hold_regs+modbus_rx.data[1]);
                        // s'haura d'enviar una resposta per aquesta funció
                        // el num de registres a llegir es *2 per coneixer el nº bytes a llegir

                        else

modbus_read_input_registers_rsp(mod_adr,(modbus_rx.data[3]*2),input_regs+modbus_rx.data[1]);

                    event_count++;
                }
                  break;
//************************************   Fi Funcio 3  *********************************//

            case FUNC_WRITE_SINGLE_COIL:
                if(modbus_rx.data[0] || modbus_rx.data[3] || modbus_rx.data[1] > 8)
                    modbus_exception_rsp(mod_adr,modbus_rx.func,ILLEGAL_DATA_ADDRESS);

                else if(modbus_rx.data[2] != 0xFF && modbus_rx.data[2] != 0x00)
                    modbus_exception_rsp(mod_adr,modbus_rx.func,ILLEGAL_DATA_VALUE);

                else
                {
```

```
                    if(modbus_rx.data[2] == 0xFF)
                        bit_set(coils,7-modbus_rx.data[1]);
                    else
                        bit_clear(coils,7-modbus_rx.data[1]);


modbus_write_single_coil_rsp(mod_adr,modbus_rx.data[1],((int16)(modbus_rx.data[2]))<<8);

                    event_count++;
                }
                break;

//***********************************   Funcio Funcio 6  *******************************//
            case FUNC_WRITE_SINGLE_REGISTER:
                if(modbus_rx.data[0] || modbus_rx.data[1] >= 8)
                    modbus_exception_rsp(mod_adr,modbus_rx.func,ILLEGAL_DATA_ADDRESS);

                else
                {
                    //Es guarda a la taula Hold regs Posició Mdb_rx.data1 els valors
                    // de mdb.rx_data3 i mdb.rx_data2
                    hold_regs[modbus_rx.data[1]] = make16(modbus_rx.data[3],modbus_rx.data[2]);

modbus_write_single_register_rsp(mod_adr,make16(modbus_rx.data[0],modbus_rx.data[1]),make16(modbus_rx.dat
a[2],modbus_rx.data[3]));
                }
                break;
//***********************************   Fi Funcio 6  *******************************//

            case FUNC_WRITE_MULTIPLE_COILS:
                if(modbus_rx.data[0] || modbus_rx.data[2] || modbus_rx.data[1] >= 8 ||
modbus_rx.data[3]+modbus_rx.data[1] > 8)
                    modbus_exception_rsp(mod_adr,modbus_rx.func,ILLEGAL_DATA_ADDRESS);
                else
                {
                    int i,j;

                    modbus_rx.data[5] = swap_bits(modbus_rx.data[5]);

                    for(i=modbus_rx.data[1],j=0; i < modbus_rx.data[1]+modbus_rx.data[3]; ++i,++j)
                    {
                        if(bit_test(modbus_rx.data[5],j))
                        bit_set(coils,7-i);
                        else
                        bit_clear(coils,7-i);
                    }
                    modbus_write_multiple_coils_rsp(mod_adr,
make16(modbus_rx.data[0],modbus_rx.data[1]),make16(modbus_rx.data[2],modbus_rx.data[3]));
                    event_count++;
                }
                break;

            case FUNC_WRITE_MULTIPLE_REGISTERS:
                if(modbus_rx.data[0] || modbus_rx.data[2] || modbus_rx.data[1] >= 8 ||
modbus_rx.data[3]+modbus_rx.data[1] > 8)
                    modbus_exception_rsp(mod_adr,modbus_rx.func,ILLEGAL_DATA_ADDRESS);
                else
                {
                    int i,j;

                    for(i=0,j=5; i < modbus_rx.data[4]/2; ++i,j+=2)
                        hold_regs[i+modbus_rx.data[1]] = make16(modbus_rx.data[j+1],modbus_rx.data[j]);

                    modbus_write_multiple_registers_rsp(mod_adr,
                    make16(modbus_rx.data[0],modbus_rx.data[1]),
                    make16(modbus_rx.data[2],modbus_rx.data[3]));

                    event_count++;
                }
                break;

            default:
                modbus_exception_rsp(mod_adr,modbus_rx.func,ILLEGAL_FUNCTION);

        }
    }
}
        ////////////////////////////////////////////////////////////////////////////////
        //////////////////////////////////////////////////////////////////////////////*//
        //*************************************************************************TFC.3_V4//
        ////////////////////////////////////////////////////////////////////////////////
```